MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

Working Paper 110 August 1975

CONTROL, MULTIPLE DESCRIPTION, AND PURPOSE IN THE VISUAL PERCEPTION OF COMPLEX SCENES: A PROGRESS REPORT

Michael R. Dunlavey

Abstract

This memo describes a vision program for recognizing simple furniture comprising assemblies of blocks, in which the same item may be composed in diverse ways. As such, it is concerned with three theoretical issues, perceptual processing, supression of unwanted detail, and segregation and interconnection of information.

The program's perceptual processing relies on an elaborate, redundant, alterable model of the scene rather than on any clever process structure. This approach aids the interpretation of incomplete, ambiguous portions of the scene as well as simplifies the program. The model is capable of quantitative as well as qualitative alteration, by a constraint-propogation system and a system of frame-shift demons.

The hierarchical nature of the scene - assemblies of assemblies of blocks - is reflected as hierarchy in the model. Each assembly is represented as having an external aspect, by which it relates to surrounding assemblies, and an internal aspect, listing the parts and relationships composing it. This imposes a natural supression of detail.

In addition to the vertical layering of the model there are horizontal subdivisions adapted for different computational purposes. There is a 2D section representing the image, a 3D section representing the shape, and a stability section representing the physical forces and moments acting upon each unit. Each of the sections can be used through any of several indirect reference frames corresponding to different spatial viewpoints. Many computations on the model, such as stability analysis, spatial relationships, and visual matching, are greatly simplified by first selecting the proper spatial viewpoints.

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-75-C-0643.

1.0 Introduction

Scene analysis is the automatic recognition and description of the semantic contents of a visual image. Work in this field has acheived a moderate level of competence for simple scenes, usually consisting of a few blocks or other polyhedral solids piled on a table (Winston). This paper describes progress on a scene analysis program for block assemblies which are just complex enough to to demand a certain amount of "chunking" in the machine's memory of the scene.

The kind of assembly I am concerned with is the following:

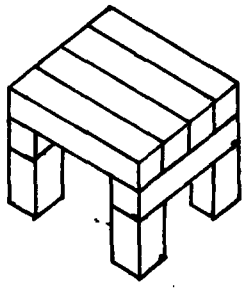


Figure 1-1. A Table Made of Many Parts

This table contains ten blocks, too many to be described without intermediate subassemblies. That is, even if a vision system could look at it and see all ten blocks (one is hidden), that description would not capture its similarity to any of the following tables:

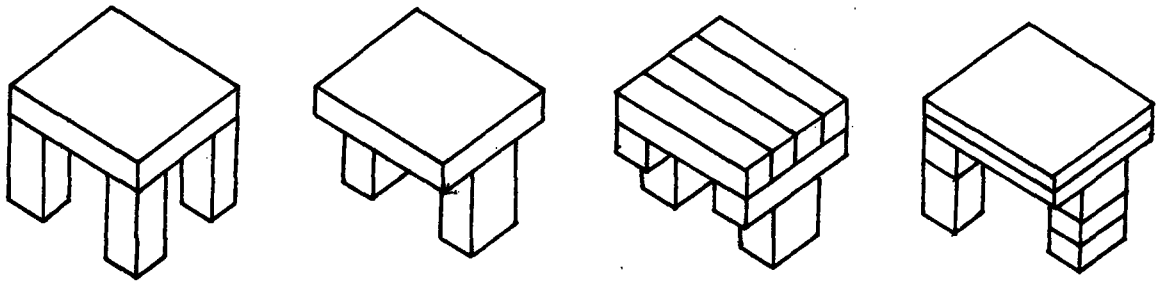


Figure 1-2. Similar Tables Made Very Differently

These tables are composed of simple nested subassemblies: rows, arches, and pedestals.

This problem was chosen to address not only problems of vision, such as the relative order of high-level and low-level concept recognition and representation of spatial knowledge, but also more general problems of epistemology, such as suppression of unwanted detail, and the interconnection of different kinds of models of the same thing.

The program's processing relies on the notion of an adjustable model of the semantic contents of the scene. The model can be created on the

basis of very little visual evidence because when it is wrong it is adjusted quantitatively or qualitatively to accomodate more data. As such, the model can be created quite early in the visual process and can act as a mechanism for conveying visual presupposition, to aid in interpreting ambiguous cues. This approach is necessitated by the accentuated ambiguity and occlusion characteristic of these complex scenes.

Suppression of unwanted detail is accomplished by modeling the scene as a roughly hierarchical collection of subassemblies. Each subassembly has an external aspect and an internal one. The external aspect represents the subassembly as a single object, acting as a first approximation to its global properties. The external aspect is the means by which the subassembly fits into the assembly of which it a part. The internal aspect describes the parts and relationships composing the subassembly.

The model of the scene, and of each of its subassemblies, is split into different sections serving different purposes. A 2D section models the image, while a 3D section models the physical shape and spatial relationships, and a Support section models the physical forces and moments acting upon the various blocks and subassemblies. These subsections of the model are in turn accessed via a system of indirect reference frames corresponding to the inherent symmetries.

The visual model is based on the idea that any subassembly has a first-order approximation in some simple known solid shape. A more sophisticated kind of global description can be built within this context, under the guidance of the viewer's purposes. For example, this scene:

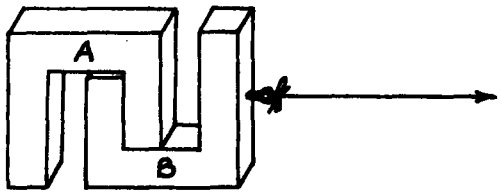


Figure 1-3. New Phenomenon - One Arch Pulling Another

could be surprising to a perceiver who does not know the relation of being "hooked-together". Understanding this example should mean building new global descriptions for A and B in which the "hooks", a previously unknown concept, are emphasized.

To summarize the following sections, first the perception of a simple pair of blocks is presented, covering most of the hypothesis-driven ideology. Then I go into hierarchical structures. Next there is an example of local-global representation-building organized by purpose. Lastly there are a speculative section on texture and a philosophical section on organization.

2.0 Hypothesis-Driven Perception for Simple Scenes

Minsky's theory of Frame Systems (Minsky) offers strong advantages for a theory of perception, and this system attempts to exploit them. It does not need to see the whole scene, but only fragments of it, and occlusion is no particular problem. At all times there is a detailed semantic model of the scene which, although it may be different from the final model, is at least consistent. The rapport between the high-level semantic model and the low-level visual data is very close, without intervening layers of computation.

When a hypothesis is wrong, it is not discarded but is transformed in an orderly way into a more correct hypothesis, avoiding wasted effort. One tenet of the frame theory is that one may use dozens of different models linked in this way. My system restricts itself to a variety of two shapes, block and wedge, so that I could concentrate on assemblies. The issues that arise with greater variety are discussed in the section on Organization.

2.1 System Outline

Input data, to keep the system simple, was restricted to the isolated vertices of a hand-coded line drawing. The data is examined by a program called the fovea, which reports only vertices contained within a movable diamond-shaped window. Each reported vertex consists of a point and the directions of its rays.

The basic operation of the system is to take vertices, one by one, and note their correspondence with points in an internal model of the scene. This process is punctuated by occasional modifications to the model to accomodate the data. Blocks appear out of nowhere, shrink and grow, sometimes split in two or turn into wedges. When the system can see no more vertices, it moves its window, looking for the far corners of objects whose dimensions are yet unverified.

The internal model for each block or wedge is a large actor (that is, a data object together with a procedure for accessing it) containing parameters for all of its corner points, principal directions, and dimensions. It thinks of itself as a complete 3-D model of the object except that the numbers stored in its points, directions, and dimensions are those of its 2-D isometric projection. Each parameter has not only its value but an authority-flag, to record whether the value is known with any certainty or is merely default. Although the block contains over a dozen different parameters, it can be adjusted as a unit due to a complex system of constraint equations for adjusting default parameters.

The process of matching vertices to points of each object is organized around two additional kinds of actors, vertex-matching-objects and vertex-

matches. Each block or wedge model contains a set of vertex-matching-objects, one for each point, which can compare a vertex to that point. If it finds a comparison, it constructs a vertex-match containing a comment about the quality or meaning of the comparison, such as "agree", "stretch" meaning the object should be stretched, "wedge" meaning it should be changed into a wedge, or "cut-after" meaning it should be divided in two. Since each vertex will usually be comparable to more than one point in the scene, the best vertex-match is chosen by sorting on the comment. The vertex-match also contains machinery to adjust or modify the model once it has been chosen. If it is to make a large change in the hypothesis, it uses tabular transfer rules to guide it in setting up the parameters in the new hypothesis.

It does not insist that hidden vertices or rays be invisible, and sometimes they're not, if the block is really going to be a wedge.

Although space does not permit greater detail, here is an outline of the contents of each type of actor:

```
Hypothesis
      Parameters
            Value
            Authority-Flag
            Immediate Hard Link
      Parameter Names - Fixed
      Parameter Names - Variable
      Rotation Rules
      Current Transformation
      Constraint Equations
      Adjustment Mechanism
      Vertex-Matching-Objects
      Suggestion Generator

Vertex-Matching-Object
      Insides of Hypothesis
      Mapping From Point and Ray Names of the Vertex
            to Point and Direction Names in the Hypothesis
      Procedure for Comparing Against Visual Vertex
      Procedure for Generating the Comment
      Prototypical Vertex-Match

Vertex-Match
      Insides of Vertex-Matching-Object
      Comment
      Procedure to Carry Out Comment
            Insides of Second Hypothesis, if any
            Transfer Rules
```

2.2 Example - Perception of a Block-Pair

Operation of the system in perceiving an aligned pair of blocks is portrayed in Figure 2-1. The initial fovea placement is provided by the user. Vertices are processed singly in any order. The first vertex, since there is no hypothesis to accept it, causes a new block hypothesis to be created, complete with default points, lengths, and directions. The second vertex establishes its x length. At this point, the parameters for the block hypothesis are the following:

```
p    = (point (3.00 2.00) nil)
px   = (point (3.50 2.50) nil)
pxy  = (point (2.25 2.50) nil)
py   = (point (1.75 2.00) nil)
pz   = (point (3.00 3.00) known)
pxz  = (point (3.50 3.50) known)
pxyz = (point (2.25 3.50) nil)
pyz  = (point (1.75 3.00) nil)
pax  = (direction 0.79 known)
pax- = (direction 3.93 known)
pay  = (direction 3.14 known)
pay- = (direction 0.00 known)
paz  = (direction 1.57 known)
paz- = (direction 4.71 known)
plx  = (length 0.71 known)
ply  = (length 1.25 nil)
plz  = (length 1.00 nil)
```

The first vertex established point pz and all the directions, and caused all the remaining points to be adjusted consistent with the first point, the directions, and the default lengths. The second vertex established point pxz, causing three other points and the x length to be adjusted. The authority-flag of the x length became "known" because the two end points were known.

The third vertex matches against the same point as the second vertex, but with the comment "row-after". A second block is hypothesized, aligned with the first and butting up against it. The fourth vertex establishes its x length and point pz. By proposing such pairs of blocks, I avoid the problem of hypothesized blocks intersecting in space. Of course this can't completely prevent it, but neither can people.

Having exhausted the visible vertices, the system asks all block or wedge hypotheses to generate suggestions. In the first block, since the y length is unknown, suggestions are generated to look for points pxyz, pyz, pxy, and py, which would establish that length. Suggested locations to look for them are based on the default value of the y length and perturbations of that value. A suggestion is taken, the fovea moved, and more vertices reported. The fifth vertex establishes point pxyz. The hypothesis is adjusted and the y length and point pyz become known. Due to

global soft equality links placed between the respective y and z lengths of the two blocks, the second block is also stretched and adjusted. At this point, the parameters for the two blocks are as follows:

	First Block		Second Block
p	= (point (3.00 2.00) nil)		(point (3.75 2.75) nil)
px	= (point (3.50 2.50) nil)		(point (4.25 3.25) nil)
pxy	= (point (1.50 2.50) nil)		(point (2.25 3.25) nil)
py	= (point (1.00 2.00) nil)		(point (1.75 2.75) nil)
pz	= (point (3.00 3.00) known)		(point (3.75 3.75) known)
pxz	= (point (3.50 3.50) known)		(point (4.25 4.25) known)
pxyz	= (point (1.50 3.50) known)		(point (2.25 4.25) nil)
pyz	= (point (1.00 3.00) known)		(point (1.75 3.75) nil)
pax	= (direction 0.79 known)		(direction 0.79 known)
pax-	= (direction 3.93 known)		(direction 3.93 known)
pay	= (direction 3.14 known)		(direction 3.14 known)
pay-	= (direction 0.00 known)		(direction 0.00 known)
paz	= (direction 1.57 known)		(direction 1.57 known)
paz-	= (direction 4.71 known)		(direction 4.71 known)
plx	= (length 0.71 known)		(length 0.71 known)
ply	= (length 2.00 known)	<- - ->	(length 2.00 nil)
plz	= (length 1.00 nil)	<- - ->	(length 1.00 nil)

After processing the remaining vertices, all of which agree (except for the T vertex, which is ignored), the uncertain z length generates another suggestion, which is taken, and the scene is finished.

2.3 Adjustable Data Structure

Each object in the data structure is an association list. For example, an hypothetical parallelogram is something like:

```
((etype . (parallelogram))
 (eguts . <external description>)
 (erules . <rule set for external description>)
       ...
       ...
       ...       )
```

A reference anywhere within an object can be denoted by a path which is a list of symbols for directing a downward search through a tree of association lists. Two functions manipulate the tree: (g* tree path) and (s* tree path value) for getting and setting respectively. Non-present names in an a-list are treated as having the value nil, and an attempt to set a non-present name results in that name being added. An example of a path is '(eguts 2d py value) which means the value of point py in the 2d portion of the external description.

The data structure for the parallelogram analogy would be something like:

```
((etype parallelogram)
 (eguts
    (2d (p    . ::p    ((value 0.0 0.0)(auth)(level . 0)))
        (px   . ::px   ((value 3.0 0.0)(auth)(level . 0)))
        (pxy  . ::pxy  ((value 3.0 2.5)(auth)(level . 0)))
        (py   . ::py   ((value 0.0 2.5)(auth)(level . 0)))
        (plx  . ::plx  ((value . 3.0)(auth)(level . 1)))
        (ply  . ::ply  ((value . 2.5)(auth)(level . 1)))
        (pax  . ::pax  ((value . 0.0)(auth)(level . 1)))
        (pax- . ::pax- ((value . 3.1415927)(auth)(level . 1)))
        (pay  . ::pay  ((value . 1.57079635)(auth)(level . 1)))
        (pay- . ::pay- ((value . 4.712389)(auth)(level . 1))))
    (t0 (q . :p)(qx . :px)(qxy . :pxy)(qy . :py)
        (qlx . :plx)(qly . :ply)
        (qax . :pax)(qax- . :pax-)(qay . :pay)(qay- . :pay-))
    (t1 (q . :px)(qx . :pxy)(qxy . :py)(qy . :p)
        (qlx . :ply)(qly . :plx)
        (qax . :pay)(qax- . :pay-)(qay . :pax-)(qay- . :pax))
    (t2 (q . :pxy)(qx . :py)(qxy . :p)(qy . :px)
        (qlx . :plx)(qly . :ply)
        (qax . :pax-)(qax- . :pax)(qay . :pay-)(qay- . :pay))
    (t3 (q . :py)(qx . :p)(qxy . :px)(qy . :pxy)
        (qlx . :ply)(qly . :plx)
        (qax . :pay-)(qax- . :pay)(qay . :pax)(qay- . :pax-)))
 (erules (list (g* foo '(eguts 2d)))
    (p (px pax- plx)(sumvec ,px ,pax- plx))
    (p (py pay- ply)(sumvec ,py ,pay- ,ply))
    (px (p pax plx)(sumvec ,p ,pax ,plx))
    (px (pxy pay- ply)(sumvec ,pxy ,pay- ,ply))
    (pxy (px pay ply)(sumvec ,px ,pay ,ply))
    (pxy (py pax plx)(sumvec ,py ,pax ,plx))
    (py (p pay ply)(sumvec ,p ,pay ,ply))
    (py (pxy pax- plx)(sumvec ,pxy ,pax- ,plx))
    (plx (p px)(vmag (vdiff ,px ,p)))
    (plx (py pxy)(vmag (vdiff ,py ,pxy)))
    (ply (p py)(vmag (vdiff ,py ,p)))
    (ply (px pxy)(vmag (vdiff ,px ,pxy)))
    (pax (p px)(theta-of (vdiff ,px ,p)))
    (pax (py pxy)(theta-of (vdiff ,pxy ,py)))
    (pay (p py)(theta-of (vdiff ,py ,p)))
    (pay (px pxy)(theta-of (vdiff ,pxy ,px)))
    (pax (pax-)(nnorm (plus pi ,pax-)))
    (pax- (pax)(nnorm (plus pi ,pax)))
    (pay (pay-)(nnorm (plus pi ,pay-)))
    (pay- (pay)(nnorm (plus pi ,pay)))))
```

The points to note about this are:

1) The external description, eguts, has more than one section, or point-of-view. 2d is the standard section, and t0 through t3 represent the same data structure from four topologically equivalent logical points of view. The funny notation involving ":" is a lisp reader hack to allow shared sublists. That is, (::x (hi there) :x) reads in as ((hi there)(hi there)) where the two (hi there)'s are eq. The "x" is just a dummy variable used at read time. So the paths (eguts 2d p) and (eguts t1 qy) reference the same memory structure.

2) There are sets of rules sprinkled around the data structure, such as erules. A rule set begins with a form which can compute the subtrees in which the rule set applies. Each rule is of the form

(destination sources form)

such as

(p (px pax- plx)(sumvec ,px ,pax- ,ply))

which says that point p can be computed from point px, direction pax-, and length plx, by eval'ing the form (sumvec ,px ,pax- ,ply) where ,px is the same as (g* foo '(px value)) where foo is the reference subtree.

3) The data structure is adjusted whenever some part of it changes. The adjustment process is to find all (subtree, rule) pairs and execute them according to the following criteria:

a) in the current update wave, there is a list of parameters which have been changed so far.

b) each (subtree, rule) pair has applicability and rank. Among the applicable rules, some highest ranking one is executed, then the applicability of all is recomputed. This repeats until no more rules are applicable.

c) A rule is applicable if one of its sources has been changed and its destination has not, and the minimum level of its sources is not less than the level of its destination, where points have level 0, lengths and directions level 1, and anything whose auth is "known" is considered to have level 2. After application, the auth of the destination becomes the minimum authority of the sources.

d) The rank of a rule is a function of the respective levels of its destination and sources, according to the formula:

rank = 2 x (min level of sources) + (level of dest)

This means that changedness will propogate

from lengths to lengths, then
from lengths to points, then
from points to lengths, then
from points to points.

This mechanism seems very ad-hoc, but it serves diverse purposes at least within this system. It also sounds incredibly complicated, but in principle it's a highly parallel operation, doing the bare minimum of actual computation on the data structure, so it need not be slow.

The purpose of this updating scheme is to allow that whenever information is dropped into one part of the data structure, its ramifications are felt throughout, or at least for some distance. In past systems (notably Sketchpad) (Sutherland), iterative relaxation was used to achieve this end, because no distinction was made between more and less important parameters. In this system, that distinction imposes an ordering on updating rules so they need only be run once. An additional shortcoming of relaxation has also been avoided, namely the reliance on numerical error. This allows some parameters to contain symbolic rather than numeric values. The notion of symbolic error is important within the theory of frame systems, but the notion of numerical relaxation is not.

In fact, the notion of a parameter having a single value largely takes the place of the idea of a fixed relation. For example, in talking about blocks, I want to say:

in viewpoint t1, the r2 relationship of block b1 to block b2 is "overhang"

In a relational data base this could be said:

(t1 r2 b1 b2 overhang)

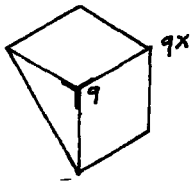
where in my system it might be said:

(irels (t1 (b1 (r2 (b1 (value . overhang))))))

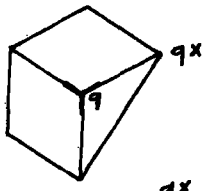
the difference being that changing the value, from "overhang" to say "align", makes the old value disappear. This trivial ability is very cumbersome to achieve in more logic-oriented representations unless one can put properties on statements (as was attempted in Conniver).

If the block hypothesis is to be changed to a wedge, or block, an analogous process takes place, where there are essentially two cases to consider. If one expected to see the lower arrow vertex of the block, then one could see either

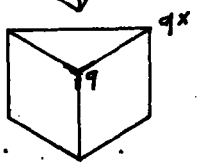
that,



or that,



or that.



Since a block hypothesis is really a parallelepiped hypothesis and is not particularly disposed toward right parallelepipeds, these three wedge cases suffice. No matter which vertex is actually seen, by selecting the right logical point of view it can be reduced to one of these three cases.

2.4 Point-of-View Shift

A program for aggregating blocks into pairs has to take account of each of the twenty-four possible axes along which the second may approach the first. This is an instance of the general problem of how to handle symmetry.

I solved the problem by letting the parameters of a block be referenced by two sets of names, fixed and variable. The fixed names all begin with the letter "p" and the variable names with the letter "q". The contents of the variable names can be a permutation of the fixed names according to an appropriate transformation. This effectively shifts the logical viewpoint that the program has of the block. Using this technique, the program can always assume that the second block lines up with the first along each one's X axis. For wedges, there is a little more complexity because there are three classes of viewpoints instead of just one.

To illustrate, in the following parallelogram the symbolic origin has been shifted one corner to the right.

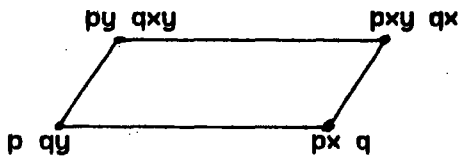


Figure 2-2. Shift of Symbolic Reference System

This technique is enormously useful, and shows up in various forms throughout this thesis.

This is another aspect of Minsky's frames theory. The programs that deal with blocks represent knowledge that is invariant under change of viewpoint. So they refer to the parameters of the block from a high level, through a level of indirect reference. Special constriants on individual parameters are not invariant and therefore refer directly to them.

2.5 Related Work

This system is most similar in spirit to those of Roberts (Roberts) and Grape (Grape) in that the problem is to find a model that can be verified using fragmentary data.

Roberts' system was the pioneering work in machine perception of solids, and my system studies only a subset of his domain. His work was not extendible because his description of each object consisted of a homogeneous transform of a prototype solid. It could only make analog, not structural adjustments to its hypotheses, nor could it represent qualitative constraints, nor could it represent imagined intermediate results or use them to guide its processing.

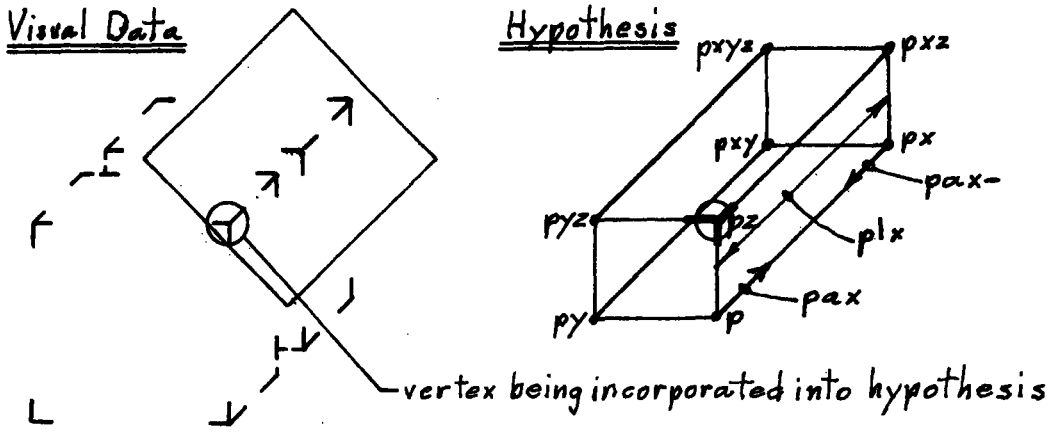
Grape's is a more serious system than mine, and he has solved technical problems associated with using real image data. However, we are both addressing the same organizational issues and my system embodies significantly different answers. His system relies on the notion of standard views of objects rather than on whole ojects. A standard view is recognized on the basis of a signature (my word) of connected line segments and associated vertices. In other words, a signature must be a large enough subset of a standard view to distinguish it from other standard views, because a mistaken choice cannot be changed. This is not a picayune objection, but is characteristic of the content-free syntactic approach to perception. To make such a system see more types of objects, one must assemble a catalogue of subsets of standard views which grows much faster than the catalogue of object types.

My system differs from Grape's in that the model is much closer to being an actual 3-D model of the scene, stating what kinds of objects are present, simple relations among them, and their precise dimensions in the image (despite occlusion). It can be more bold about hypothesizing the identity of objects on vertex evidence alone because there is no penalty for guessing wrong, as the hypothesis can be changed.

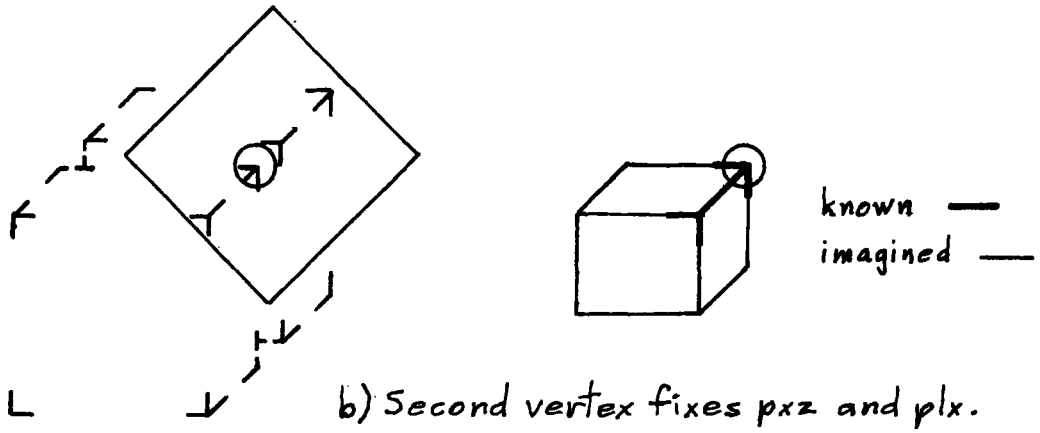
All three systems suffer from a poorly chosen vocabulary of basic models, namely blocks and wedges. Without going into detail, the problem is that these objects are effectively "defined" in terms of their lines and vertices, and these definitions are too complex. The addition of more basic shapes so defined would make it more difficult for a perception system to discriminate among them, even if it is able to change its mind. A solution for this, as previously mentioned, is to use descriptions

organized around the notions of axis and section. (Agin) (Hollerbach)

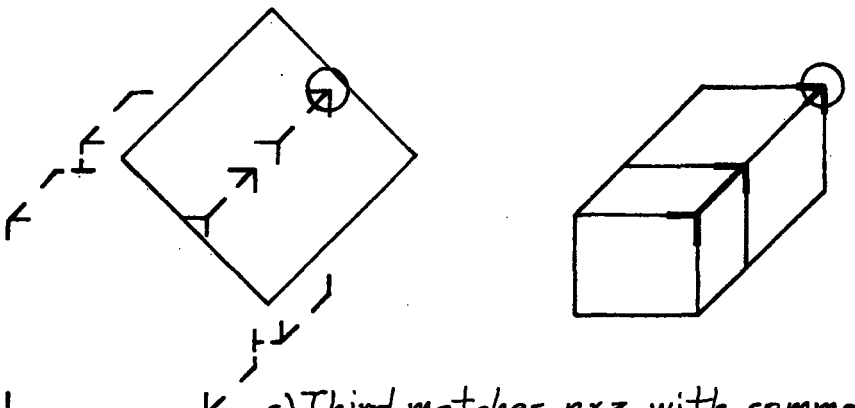
A recent paper by Kuipers (Kuipers) discusses these same issues, concentrating on formalizing the control aspects of frames for recognizing blocks and wedges on the basis of vertex evidence.



a) First vertex triggers a block hypothesis

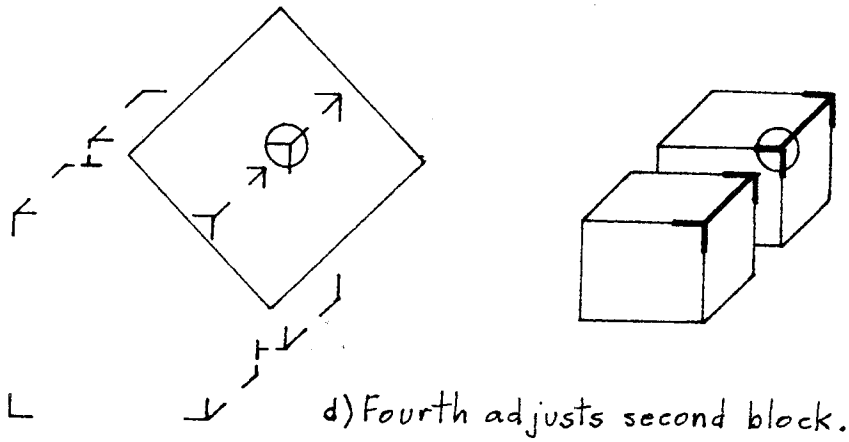


b) Second vertex fixes pxz and plx.

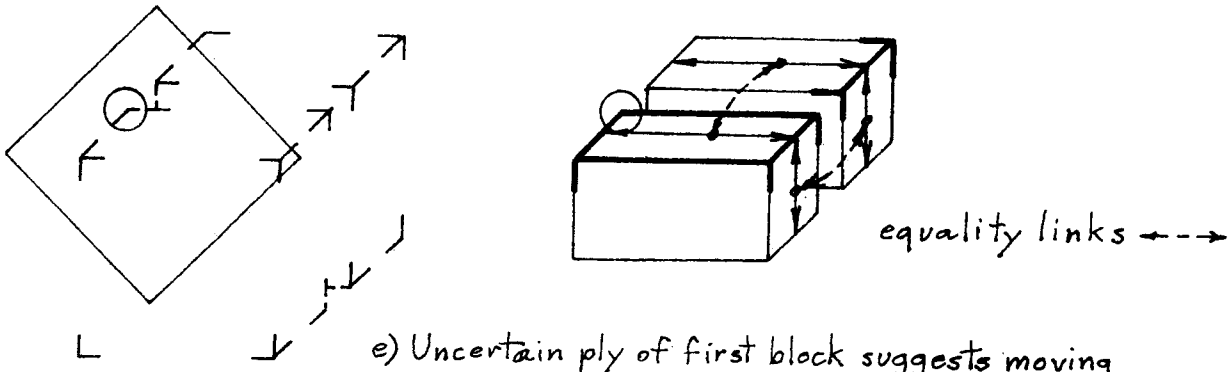


c) Third matches pxz with comment "row-after", triggering second block aligned with first.

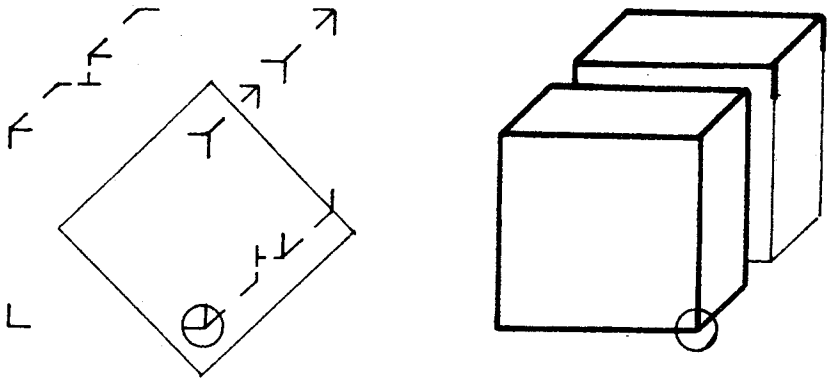
Figure 2-2. Visual Data and Hypotheses During Preception of a Block Pair



d) Fourth adjusts second block.



e) Uncertain ply of first block suggests moving fovea. Fifth vertex adjusts both blocks.



f) After three more vertices, (⊥ is ignored) fovea moves again, finding plz, and so on.

3.0 Perception of Nested Assemblies

The visual domain of nested assemblies of blocks accentuates the difficulties of blocks-world vision -- occlusion, ambiguous vertices, accidental alignments -- missing and ambiguous data in general. The approach taken is to rely heavily on an elaborate, redundant, alterable model of the scene rather than on any clever process structure.

The reader is asked to please bear with the lack of clarity of this section, as it has not been worked out as thoroughly as the previous one.

The model, or hypothesis, is a self-adjusting data structure into which morsels of information can be dropped, from the scene or from other knowledge, and it will update itself to reflect that information. An analogy is in Sutherland's Sketchpad program (Sutherland) in which a fairly complex scene containing geometric constraints could be built. That scene could then adjust itself, by a relaxation mechanism, to changes imposed by the user with his light pen. This effectively magnifies the input information. A small change in the angle of a bar of a linkage results in an entire mechanism moving. My visual models have that same property but are considerably more complex and are capable of adjustment structurally, as well as numerically.

Sketchpad allowed hierarchical descriptions by defining common subassemblies. Instances could be connected to contextual objects by means of distinguished terminal points, so that as far as the context could see, an instance consisted only of a rigid set of terminal points. My philosophy was that a subassembly instance should appear to the context like some familiar simple object, with further elaboration only when needed.

The description for an assembly can be broken down vertically into the external and internal parts, and horizontally into the 2D, 3D, and stability theory descriptions.

Within this domain, there is a simple answer to the question of how to represent the ways in which each kind of subassembly relates to its context. One thinks of each subassembly externally as having the shape of a single simple object of a kind already known, like a block or wedge, and internally as a network of parts and relations.

Each table can be described as an arch, i. e. a block-shaped top, and two block-shaped supporting subassemblies. To see such a table then is to build such a description to agree with the visual evidence.

There are nine principle ideas embodied in the structure of hypotheses:

1) Data Structure -- Each object, like a block or wedge, is represented by a redundant collection of parameters for its various points, dimensions,

etc. Each parameter can hold a value and an authority flag indicating "known" or "default". Each parameter has a fixed name by which it can be referenced. Parameters are grouped into two levels of invariance - directions and lengths are invariant under rigid translation, and points are not.

2) Variable Naming System -- There is also a group of variable names for parameters. By permuting the parameters to which these names refer, the same object can be thought of from different logical points of view.

3) Updating Links -- Each object contains a set of active links between its parameters such that a change made to one parameter can affect all. Some of these links come built-in, and others are added to carry the influence of context. These links refer to the parameters either directly, through the fixed names, or through the variable names qualified by a permutation. The execution order of these links is partial, those triggered by or affecting upper level parameters having priority. This priority obviates the need for iterative relaxation.

4) Each object has access to a suggestion generator which works by seeing which high-level parameters are unknown, which visible parameters would determine them, and where to look for them. Suggestions are passed to the visual input device, in this case the fovea.

5) Updating links may be added to carry information between related object instances, for example to convey the expectation that adjacent objects have similar dimensions or height.

6) Assemblies of objects have two halves, the external and internal descriptions. The external descriptions is the same as a simple object of some known type. The internal description is a configuration of objects. A set of special updating links, the hierarchy links, connects the parameters of the external description to some of the parameters of the internal description. The intention is that relations outside the assembly act upon it primarily through its external description which functions as a first approximation of its global properties.

7) The internal description of an assembly contains relationships among the components. A relationship has not so much a truth value as a descriptive value or qualifier. The value of a relationship is periodically updated and tested for unacceptability. Some of these relationships are qualified by a particular logical point of view.

8) The entire internal contents of an assembly, including the hierarchy links, can be replaced by another configuration having similar external form. This is done upon suggestion of a frame-shift demon which is cued by the relationship values of the internal description.

9) Each unit is not just one collection of parameters and links, but three (in this system) representing different ways of thinking about the object.

One represents the two-dimensional image, one represents the three-dimensional shape, and one represents the support points, moments, and forces of the objects. These "theory slices" are interconnected by updating links which convey the 2D-3D transform, occlusion expectations, and height assumptions.

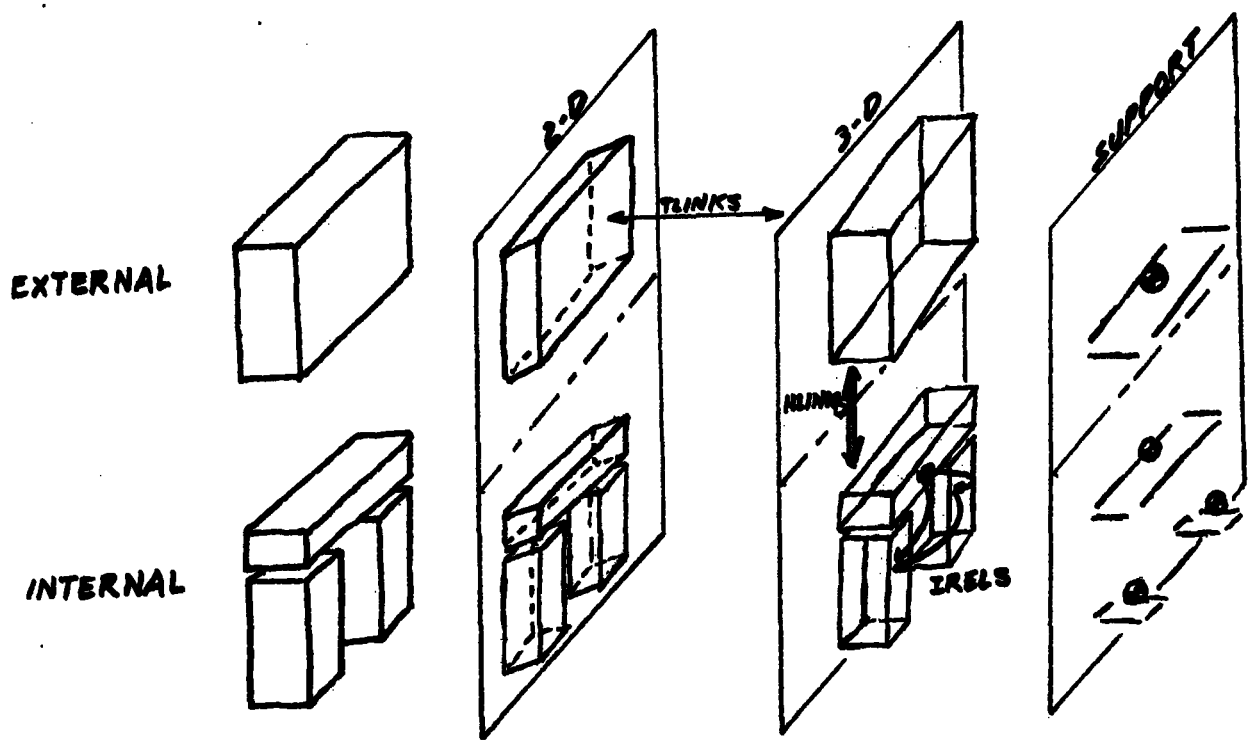
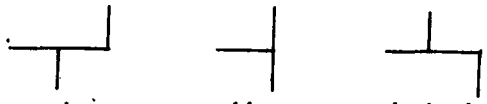


Figure 3-1. Rough Diagram of External-Internal Description

```
(ushape-block                          ;typical assembly description
   (etype block)                          ;external type is a block
   (eguts                                 ;external descriptions
      (2d
         (t0 (p ...)(px ...) ...)
         (t1 ...)
         ...)
      (3d
         (t0 ...)
         (t1 ...)
         ...)
      (support (center ...)        ;in the stability theory
               (weight ...)                ;intrinsic weight & moment
               (moment ...)
               (w1 ...)                       ;weight on & location of
               (r1 ...)                  ;first support point
               (w2 ...)                       ;likewise for second point
               (r2 ...)
               (total-weight ...)             ;due to things above also
               (total-moment ...)))
   (tlinks                                ;transfer links between theories
      (2d-3d ...)                            ;for the external description
      (3d-2d ...)
      (3d-support ...)
      (support-3d ...))
   (itype ushape-block)                   ;internal type is "ushape"
   (iguts                                 ;internal description components
      (b1 (etype block) ...)                 ;are three objects of external
      (b2 (etype block) ...)                 ;type "block"
      (b3 (etype block) ...))
   (irels                                 ;internal descriptive relations
      (2d ...)                               ;in each of the theories
      (3d (b1 (rel1 (b2 align))
              (rel2 (b2 overhang)))
          (b2)
          (b3 (rel1 (b2 overhang))
              (rel2 (b2 align))))
      (support
         (b1 ...)
         (b2 (s1 b1)                          ;supporter at first point
             (s11 (at (u v w)))               ;location of first point
             (s2 b3)                          ;likewise for second point
             (s12 (at (u v w))))
         (b3 ...)))
   (hlinks                                ;hierarchical links
      (2d ...)                               ;in each of the theories
      (3d ((p)(b1 eguts p))                  ;between external and internal
          ((px)(b3 eguts px))                ;descriptions
          ...)
      (support ...)))
```

The spatial relations, REL1 and REL2, are the keys to the identity of the internal description. They refer to the relative arrangement of corners of blocks:

REL1= overhang align indent



REL2= overhang align indent

Each relation is qualified by the transformation in which it applies. An external program can rename them in such a way as to be relative to any external framework, so as to match against another description.

On the basis of these relations, there can be transformations that change from one structure to another, like the following row-to-ushape transform:

```
(rul                                ;typical transition demon
   (source row-block)                  ;from structure "row"
   (state ((iguts b1 exists) t)        ;can apply if b1 and b2 are seen
          ((iguts b2 exists) t))
   (complaint ((irels 3d t0 b1 rel2 b2)    ;if this relation which should be
               (align overhang)))          ;"align" is "overhang"
   (destination ushape-block)          ;suggest changing to "ushape"
   (map (b2 . b2)(b1 . b1)))           ;using this mapping of parts
```

A straightforward scenario for perceiving a two-dimensional analogy of the table would be:

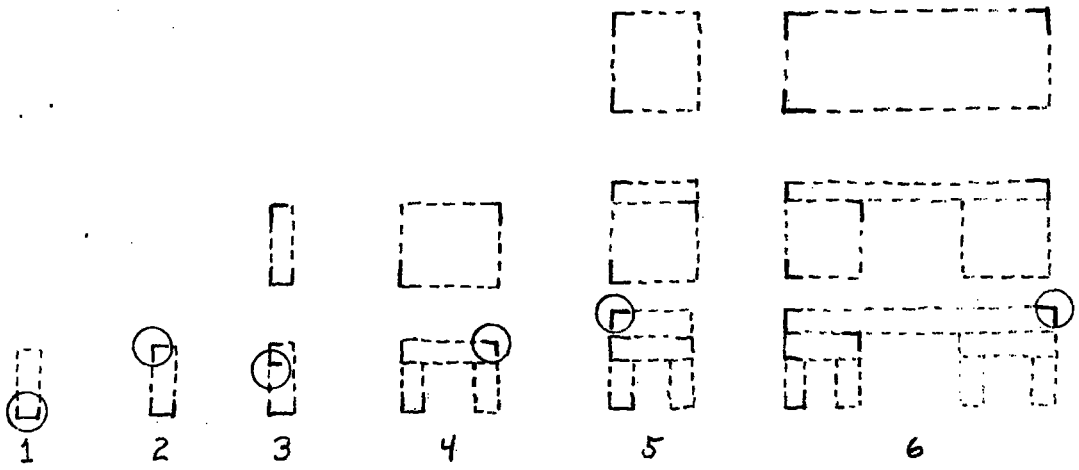


Figure 3-2. Perception of Table in Six Steps - 2D Analogy

The problem with this sequence is that it's too ideal. If vertices are seen in another order, the program can jump to conclusions that are so wrong as not to contribute directly to further processing. For example, in this 3D example from the table, the front board on top and the right-front leg are erroneously grouped into an arch.

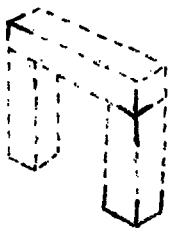


Figure 3-3. Erroneous Front Arch of Table

The error is discovered when the right-front leg is found to belong to another arch.

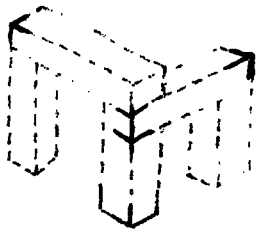


Figure 3-4. Same Arch After Right Leg Turns Into X+ Arch

The error is discovered in three steps, 1) the right front leg (which is a block) becomes an arch (which is a block externally), but its external dimension increases in the X+ direction. 2) Next the first arch, of which the second arch is now a support, reevaluates the spatial relationships among its parts, and finds an excessive error of alignment at the X+ face. 3) In the collection of transformations among known assemblies, there are none which can adjust to this error, so doubt is cast on the entire first arch, but not on its constituents which were seen.

3.1 Cross-Theory Connections

The 3D theory slice is the focus of a description. The 2D slice acts primarily to pass information into it, and the support slice refers to it.

The 2D to 3D transfer links attempt to enforce the constraint that the height of the base of each object is determined by its supports. There is also a 2D to 3D transform. There is also a marker on each 2D point saying whether or not it is definitely, maybe, or not occluded, and by what. For example, for two blocks lined up such that qax = pax, pay, or paz-, point qy of the second block is certain to be occluded if the separation between the two blocks is small, otherwise maybe. If the front block stretches especially far to the right, point qx of the back block is maybe occluded.

Each object's support description consists of two orthogonal 2D support descriptions. The contact points are functions of how the object has been hypothesized to mate with its support in the appropriate plane of the 3D description. The support slice contains two descriptions, one in the xz plane and one in the yz plane. The location of each support point in each plane is a function of the rel1 and rel2 relations in the corresponding plane of the 3D slice.

The levels of these cross-slice connections, for the 3D-support pair of slices will be a function of the orientation of the 3D assembly. The 2D 3D occlusion connections will be a function of the absolute point names and the absolute pairwise axes.

3.2 Occlusion Theory

Each visible vertex has a "visibility" flag, which takes values "yes", "maybe", or "no". Only "yes" vertices get matched against seen vertices, followed by "maybe" vertices.

This is essential, because although the number of possible matches is small for a simple wire-frame figure:

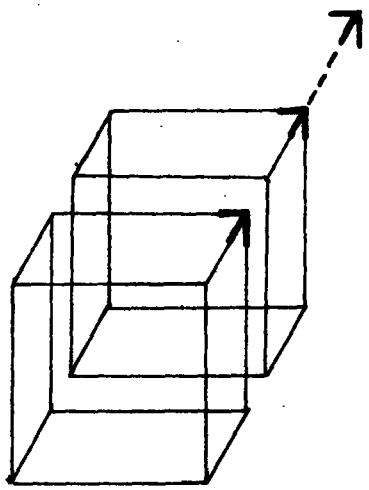


Figure 3-5. Occlusion Masking not Needed on Simple Scene

it may go as n to cube root of n for an n-block wire frame figure.

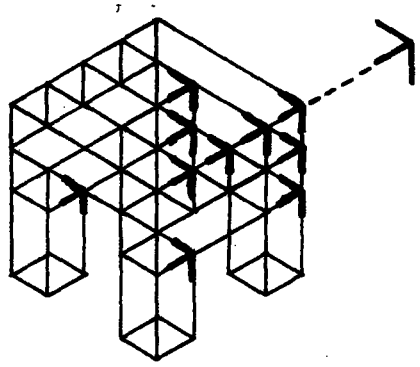


Figure 3-6. Complex Scene Without Occlusion Masking - Increased Ambiguity in Vertex Matching

Occlusion information comes from the 2D theory slice. If a pair of blocks

forms a row, certain points of the occluded block are marked "yes", and some "maybe". A "yes" or "maybe" mark may be changed to "maybe" or "no" by more evidence. The flag moves down hierarchy links between points, and if a sub-block so receives a "no" or a "maybe" on one of its points, the whole sub-block receives a "no" or "maybe".

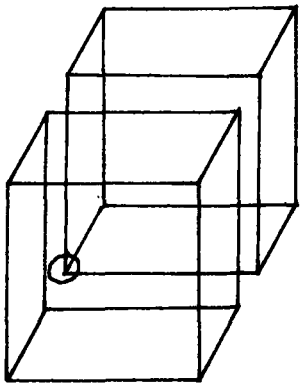


Figure 3-7. Definitely Occluded Point in Row-Block

For the align-overhang relationship of blocks (l-shape), there are six configurations of greater occlusion than found in a row. These simple rules work fairly well for the sample scenes.

4.0 Learning a Global Description

I believe that a child at play, a scientist, or an inventor is learning concepts which help him to solve problems. The issues already discussed, local-global description, purpose, and perception, shed light on this process.

In this example, a learner, who is reasonably familiar with assemblies of blocks, learns the concept of a "hook", not by being told it, or by seeing repeated instances, but by inventing it to describe the important point of an experiment. I do not suggest how the experiment is proposed, but I do suggest how it could be understood.

I was led to seek this kind of example out of a great uncertainty as to what kinds of properties should be present in a global description. In other words, the problem is not so much finding the global description, as finding which global description.

The point of this example is that the global description is made which pertains directly to the purpose of pulling. An auxiliary point is that, to make this description, a new concept must be formulated, the "hook", along with descriptions of its form and use, and purpose in the guise of causality serves to delineate that concept.

The learner is confronted with the following scene:

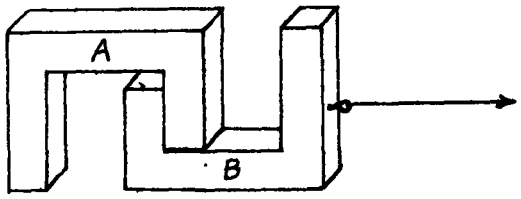
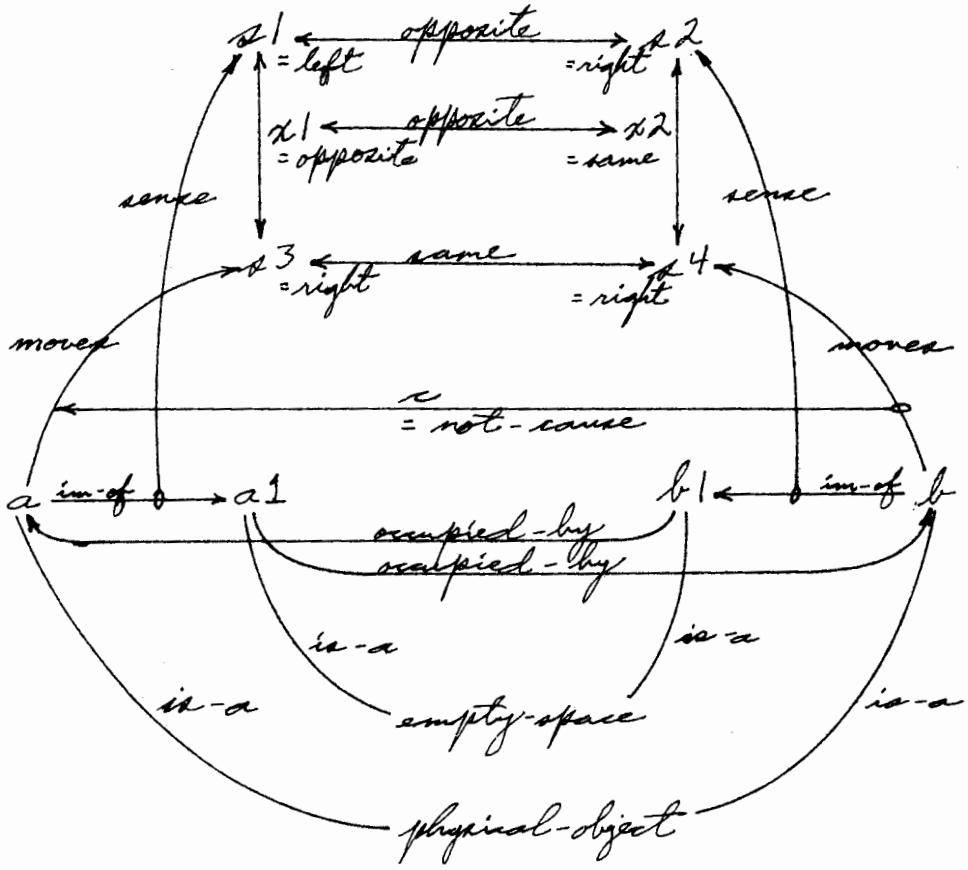


Figure 4-1. Surprise! - One Arch Can Pull Another

His perceptual process describes A and B as two rigid arches, each block-shaped, with B to the right of A. He knows that B isn't really to the right of A, but he doesn't know how else to describe it. This description matches the "separating" entry point of the two-object-motion frame system shown in Figure 4-2, making him expect that B's motion will not cause A to move. Since B's motion does cause A to move, which matches "pulling", he is surprised.

```
(DEF TWO-OBJECT-MOTION-SYSTEM
   (POSITIVES A A1 (A IMM-OF A1)((A IMM-OF A1) SENSE S1)
                   (A1 SPACE-FOR B)
              B B1 (B IMM-OF B1)((B IMM-OF B1) SENSE S2)
                   (B1 SPACE-FOR A)
              (A MOVES S3)(B MOVES S4)
              ((B MOVES S4) C (A MOVES S3)))
   (VARIABLES (S1 (LEFT RIGHT))
              (S2 (LEFT RIGHT))
              (S3 (LEFT RIGHT))
              (S4 (LEFT RIGHT))
              (X1 (SAME OPPOSITE))
              (X2 (SAME OPPOSITE))
              (C (CAUSE NOT-CAUSE)))
   (CONSTRAINTS (S1 OPPOSITE S2)
                (S3 SAME S4)
                (S1 X1 S3)
                (S2 X2 S4)
                (X1 OPPOSITE X2))
   (DEF SEPARATING
      (DEFAULT (S1 LEFT)(S2 RIGHT)
               (S3 RIGHT)(S4 RIGHT)
               (X1 OPPOSITE)(X2 SAME)
               (C NOT-CAUSE)))
   (DEF PUSHING
      (DEFAULT (S1 LEFT)(S2 RIGHT)
               (S3 LEFT)(S4 LEFT)
               (X1 SAME)(X2 OPPOSITE)
               (C CAUSE)))
   (DEF PULLING
      (DEFAULT (S1 LEFT)(S2 RIGHT)
               (S3 RIGHT)(S4 RIGHT)
               (X1 OPPOSITE)(X2 SAME)
               (C CAUSE)))
   (TRANSITIONS (SEPARATING (X1 SAME PUSHING)
                            (C CAUSE PULLING))
                (PUSHING (X1 OPPOSITE SEPARATING)
                         (X2 SAME PULLING))
                (PULLING (C NOT-CAUSE SEPARATING)
                         (X2 OPPOSITE PUSHING))))
```

Figure 4-2a.



c =	x1= opposite, x2= same	x1= same, x2= opposite
cause	pulling	pushing
not-cause	separating	

Figure 4-2b.

The learner is not only surprised, but he recognizes this as a useful phenomenon because it fills a gap in his index of methods for making things move:

push -- (shoving algorithm)
needs nothing extra
pull -- (gluing algorithm)
needs glue, messy, irreversible
-- (stringing algorithm)
needs string and little holes in objects
-- (new algorithm)*
depends only on shape

Next he does a causal analysis. That is, he makes proofs (See Figure 4-3) in which the causality is explicitly represented as links in the description (a la Schank). (Schank) First the learner proves that the event was possible by looking at the next lower level of detail. He compares this proof with a recalled proof of why the objects should separate. The difference lies in the relationship of the two blocks forming the contact interface, which is opposite to what would normally be expected for contact among wholes.

Finally he makes a proper generalization of this difference. The difference is divided so as to isolate the portions which belong to each side. Then, on each side, is a description that says that an extreme block, rigidly fixed to its parent assembly, has space on the face opposite to its external face, to be occupied by a block from another assembly. This statement forms the new concept corresponding to what humans call a "hook". (See Figure 4-4). If two objects have hooks, then in that sense they are the same as these two arches, and should therefore behave the same.

Note that the hook description makes explicit reference, via context variables, to the other hook with which it potentially mates. A context variable is an explicit reference to some item in the object's spatial context. It is variable because it can be filled either by a neighboring object or the space that object would occupy.

As always, the problem in learning is not how to remember an event but how to make the right generalized statement of what happened. (also how to find the right things to learn.) The right kind of statement to remember is one which captures the principle or crucial feature of the event as separate from the irrelevant details. This scenario shows how the principle of an event can be found by analyzing it in terms of causal models of behavior, in which "purpose" takes the form of a chain of causation.

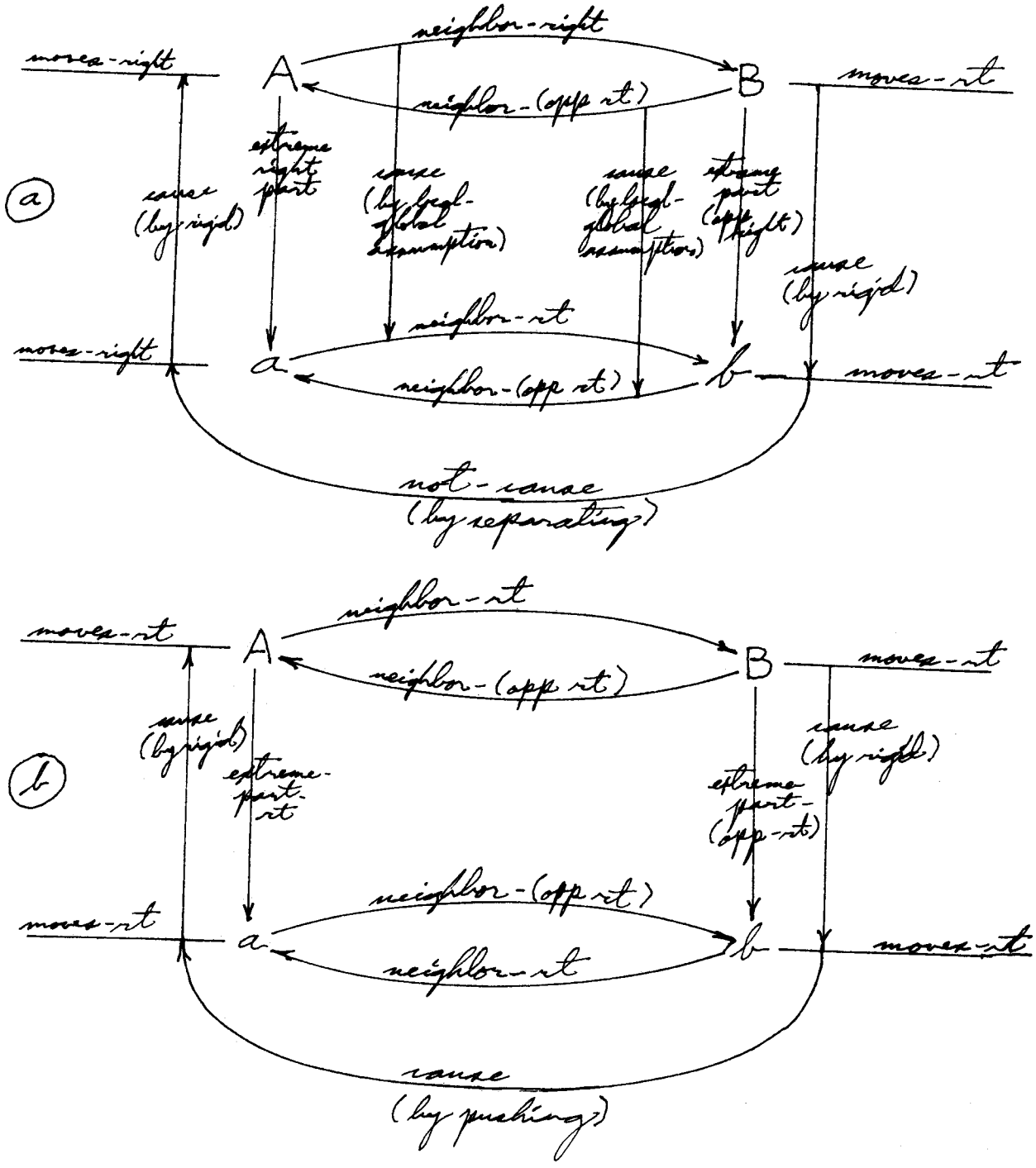


Figure 4-3. a) Proof of Separating
b) Proof of Pulling

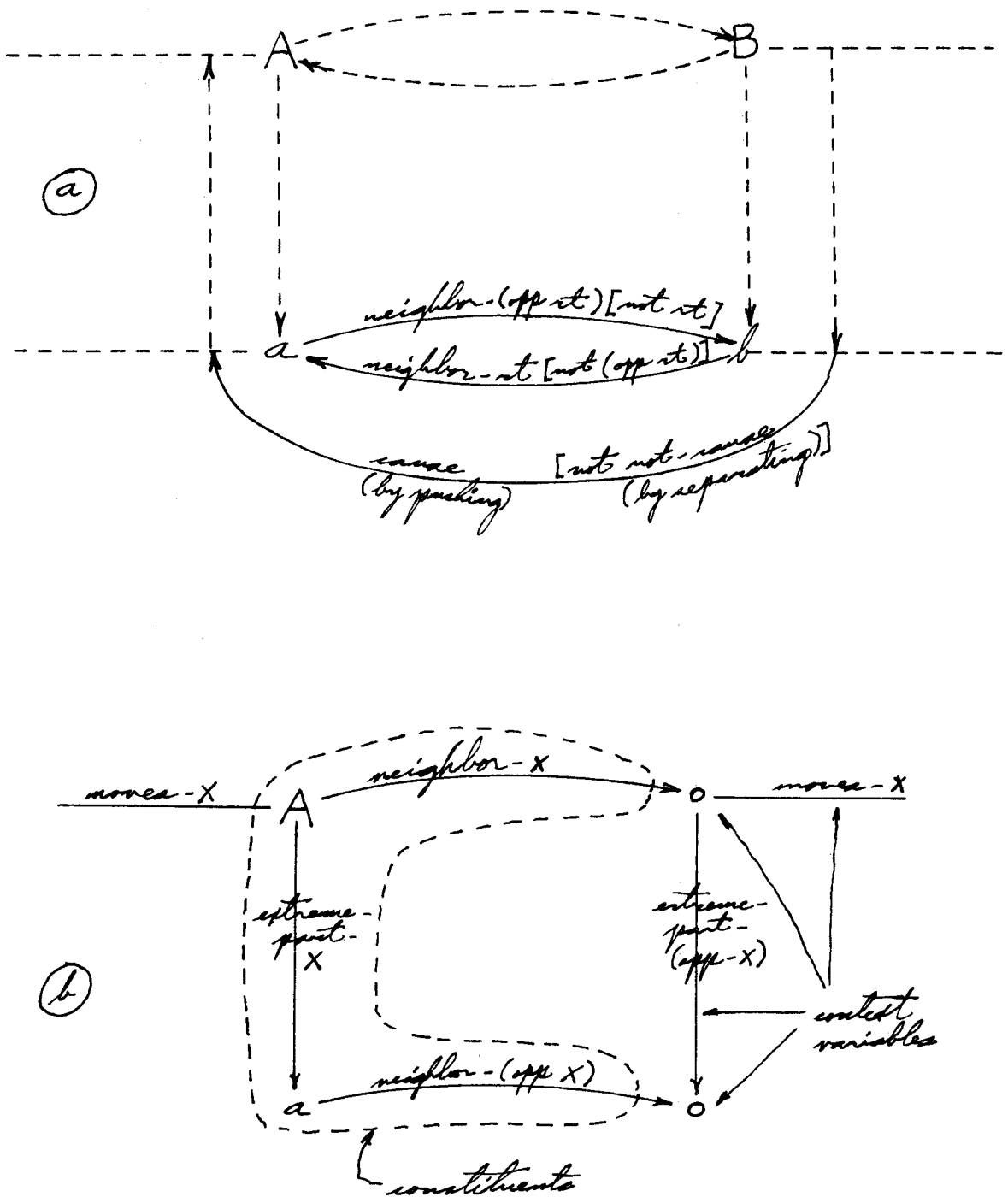


Figure 4-4. a) Comparison of Proofs
b) Isolation of "Hook" Concept

This example of learning builds from Winston's program (Winston) in two theoretically important respects:

Molecularism -- Winston's program learns on the basis of feature differences which are more or less obvious. Blocks, wedges, and relations are considered atomic units without any fine structure. More realistic descriptions cannot ignore the lower details of shape, corners, edges, sides, axes, spaces, contact points, etc. In the presence of all this detail, differences are not so obvious, so purpose must be used to project the description down to a simpler one, or to emphasize the important detailed features.

Indication -- In Winston's program, a training sequence of examples serves to indicate the constituents of the concept and then to emphasize certain single features, one per example. It is inherently difficult to indicate a conjunction of differences because there usually are several differences lying around anyway, and the program has no way to know that more than one was intended. In the scenario presented here, the causal chain serves to indicate a conjunction of differences which might not otherwise seem important. It is that causal chain which indicates the parts of the "hook" and emphasizes them.

5.0 Complexity, Texture, and Figure-Ground

One direction for improvement of the system described here is to permit condensed, procedural description of <u>really</u> complex scenes. It is unreasonable to expect a perceiver to build a complete description of a brick wall, for example. A scene which tests the system's capacity is the following row of twenty-two blocks:

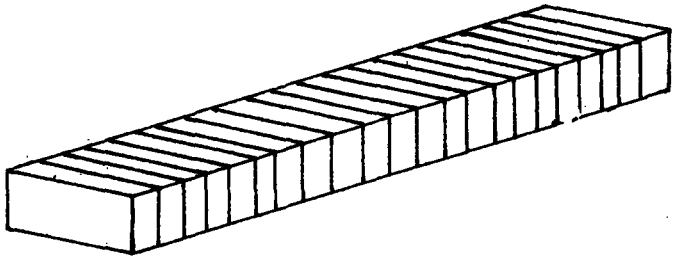


Figure 5-1. Row of Too Many Blocks

My system would attempt to describe this as an assembly of twenty-two blocks, having the overall shape of a single block. A better way to represent this would be as a <u>repetition</u> of blocks having the overall shape of a single block, something like a Fortran DO loop with end conditions. Then, only as much of it need be instantiated as is the object of attention:

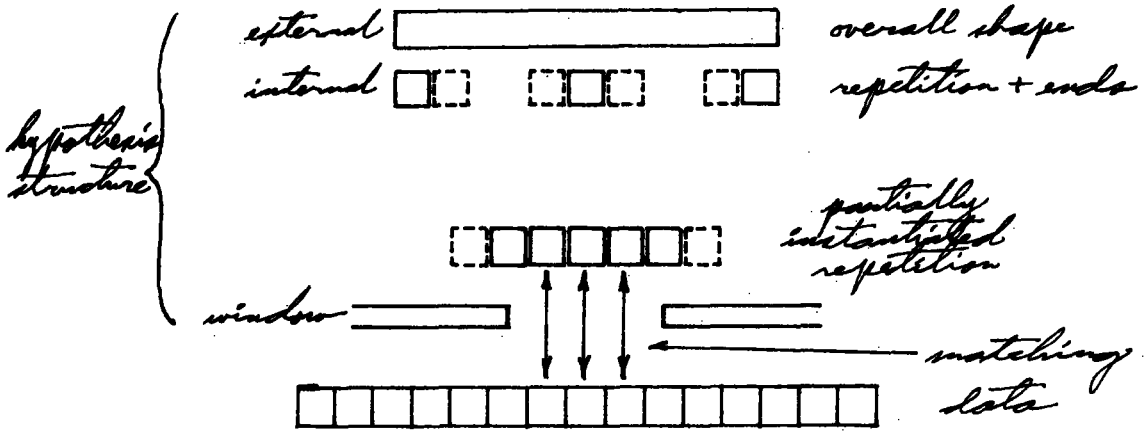


Figure 5-2. Operation of a Procedural Model

The instantiation process for procedural descriptions is not difficult theoretically, as long as interconnections can be made in a relatively uniform manner. For example, a recent thesis by (Baumgart) describes a

system called Geomed in which procedures build and manipulate rather detailed data structures, while maintaining strong global properties, like getting a torus to close on itself. This is the kind of thing I have in mind as procedurally described data structure, but it needs to be augmented with adjustability and ability to change structurally in response to the data. This will probably mean giving up the exact registration of different parts of the description.

The next question is "How could this kind of hypothesis get built?" My system would first see two or three individual blocks, hypothesize the row and its overall shape, and procede to verify it.

Recent theoretical work by Marr (Marr) on computational models of the mamalian visual cortex is relevant to this point. He would say that before one sees blocks, one sees an elongated texture patch representing the entire object, having a boundary of connected lines, where the texture consists of parallel line segments, i.e. that the initial guess at global form, or figure vs ground, arises not from the blocks, but from the lines. He proposes that simple shapes like blobs and stripes are found in this way, and that the process is recursive, so that, for example, stripes of blobs can also be found. This would explain the square vs diamond illusion:

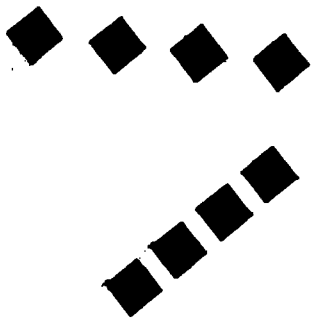


Figure 5-3. Global-Prior-to-Local Perception

in which the principal axis, part of the global form, is derived prior to description of the local entities, effecting their perception as diamonds or squares, respectively.

On the other hand, it is possible for the local perception to be prior to the global, as for example in noisy scenes like road maps. Usually on a road map, of say Massachusetts, the name of each county is printed in light-colored, widely spaced letters:

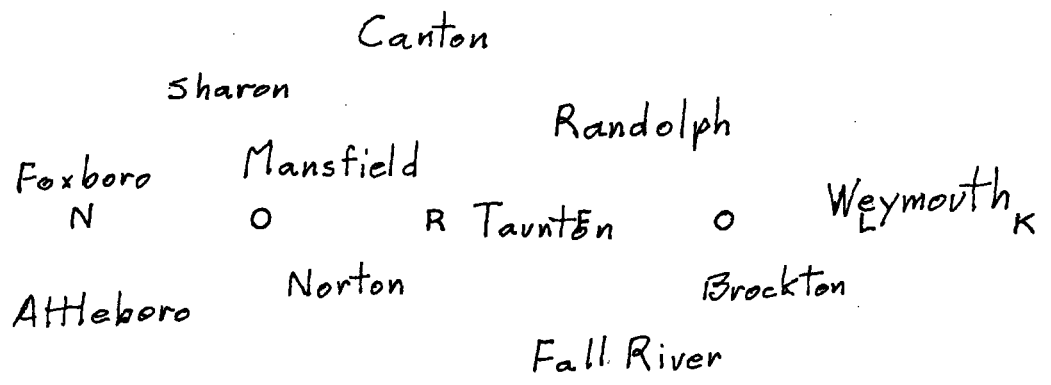


Figure 5-4. Local-Prior-to-Global Perception

To see what county it is, one has to look for one or two letters, hypothesize where the rest of the word is, and then look for the remaining letters.

5.1 The Appearance of a Bolt

The end of a threaded bolt is a good example of texture (the threads) superimposed on an overall shape. It is a good example of texture which could easily admit a procedural description.

The appearance of a small cylinder is fairly easy to characterize. It generally has one highlight, somewhat off-center. On either side of the highlight the intensity falls off smoothly to merge with the side patches. The side patch on each side reflects the immediate background, and it is light or dark accordingly. Then there is usually a shadow on the side opposite the main highlight. Surrounding all this is the background level itself. (Lozano)

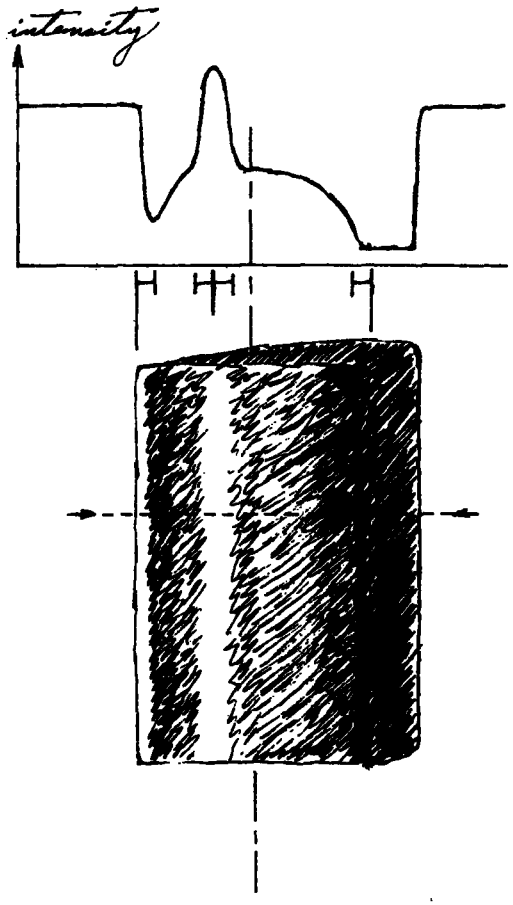


Figure 5-5. Appearance of Simple Cylinder

A model of this would contain, at the top level, an axis direction, end points, and radius for the cylinder, direction of illumination, distance to background, intensity and tilt of background. Below this would be axial offset, brightness, and width of the highlight, matte brightness of the cylinder, brightness and position of side patches, position and width of shadow. Then there would be examiners for the various features, highlights, shadow, side patches, and ends.

To extend this model to a bolt, it is necessary to represent the thread texture. Looked at finely, threads look like:

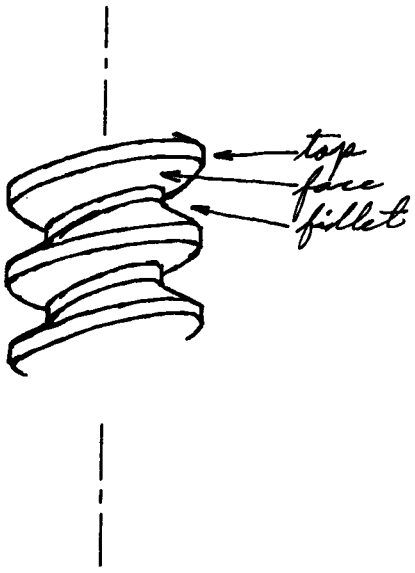


Figure 5-6. Appearance of Threads

Going across the axis, the top, face, and fillet of a thread each has an intensity profile which is similar to that of a simple cylinder. In addition, all the tops tend to look alike, and so do the faces and fillets, so that profiles parallel to the axis would also be parallel, at least locally. The threads are quite amenable to an iterative description. This iteration could move along the overall axis of the bolt. There can be local modifiers to the effect that the threads are more shiny at one end than at the other, for example.

6.0 Organization and Representation

I have said that this system is primarily top-down. By this I mean that the information at the top decides what to do as a result of each morsel of evidence. This is because the morsels themselves are too unreliable, incomplete, and ambiguous to tell us directly what top-level objects there are. But this leads the system to a strange impasse; it cannot see things which it cannot guess at the top level, and it cannot distinguish a very wide variety of things on single vertex appearance alone. This may not be a complaint against frame systems in general so much as against first attempts to build them using feeble representations.

The top-down vs. bottom-up controversy seems to be really a matter of coherence vs. flexibility. To encapsulate the argument:

bottom-up:
- local
- admits highly parallel processing
- good for detecting surprises
- flexible - reports arbitrary conjunctions of simple features
- upward ambiguity increases with variety of scenes

top-down:
- global
- builds meaningful hypotheses
- great on ambiguous input
- hard to incorporate much variety

Good straw men to compare these approaches are Waltz's system (Waltz) and mine.

Waltz's system is flexible. It can process almost any polyhedral solid, with shadows, and label all the line segments as convex, concave, or occluding, for example the following:

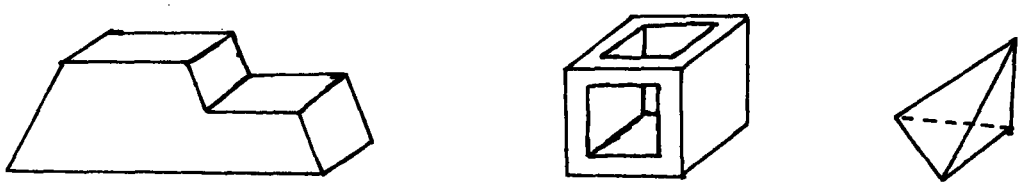


Figure 6-1. Arbitrary Polyhedra

My system wouldn't have a prayer on these shapes because a) it can't imagine them, and b) even if it could, it couldn't distinguish them on single vertex appearances.

On the other hand, when Waltz's system has processed these drawings it still doesn't know what's there. All it has is a network of vertices and

labeled lines. For example, it can't tell you what the shape is, and it doesn't realize there is a hole. Of course it wasn't intended to do those things, which merely reinforces the argument.

My system, on the other hand, likes this kind of drawing:

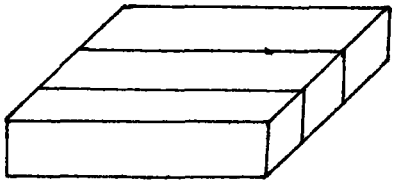


Figure 6-2. Row of Three Blocks

in which two of the blocks, though obscured and having ambiguous T and PSI vertices, are seen as clearly as the one in front, and the collection has the overall shape of one big block. The drawing can be quite incomplete because the system demands only the bare minimum of agreement with the hypothesis. However, it has trouble with a simple variant:

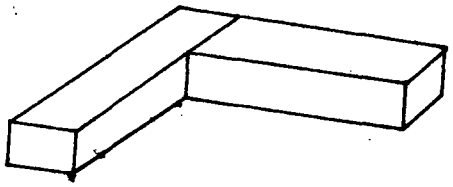


Figure 6-3. L-shape of two blocks

This configuration is seen either as a row with an anomalous member or as an incomplete ushape. It is also seen as two blocks aligned at one corner. It is not seen as a coherent l-shape because that is not in its battery of basic shapes. For example, it can't tell you where the "inside corner" is. Although l-shape could be added once one figured out how it could go into combination with larger assemblies, it is not hard to find harder examples.

These two systems represent extremes in the bottom-up vs. top-down tradeoff. However, there is a middle ground which can be gotten by a better representation, namely generalized cylinders (Agin) (Hollerbach).

A generalized cylinder has an axis, which is some line segment in space, possibly bent, and one or more cross section shapes suspended on it. Simple variations of these basic features can represent a reasonably wide variety of shapes.

The way my system is presently organized, it goes directly from vertices to blocks (and thence to larger blocks):

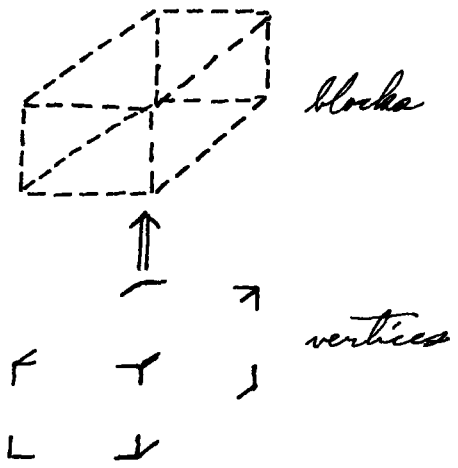


Figure 6-4. Current organization

A better way to organize it would be to use cylinders as an intermediate representation:

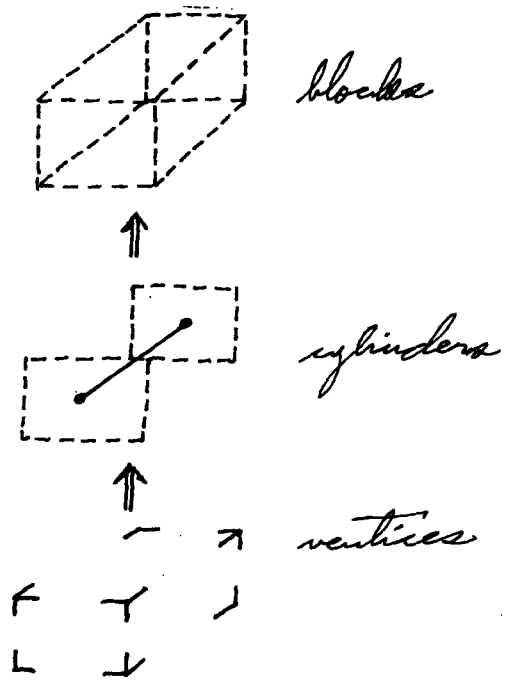


Figure 6-5. Using Cylinders as Intermediate Representation

If this were then tied together with a Waltz-like system operating on the input, it would combine the best features of all.

In all this, one should keep in mind that blocks are a highly

artificial and constrained visual world. It is a little strange compared to people vision where everyday objects can be identified on the basis of local cues alone. For example, Piaget reports (Piaget) that children who have difficulty distinguishing between a triangle and a trapezoid have no trouble on a fountain pen, spoon, scissors, flower, etc. At this level, comparing my vision system to a real one is a bit like comparing the alphabet of a Turing machine to a children's dictionary. The blocks world is overly concerned with arrangement of features. This is good because it has to be tackled. However, real vision relies more heavily on the unique content of individual micro-scenes, texture, color, shape, etc.

References

1. Agin, G., Representation and Description of Curved Objects, Memo AIM-173, Stanford Artificial Intelligence Project, October, 1972.

2. Baumgart, B., Geometric Modeling for Computer Vision, AIM-249, Computer Science dept., Stanford University, October 1974.

3. Clowes, M., unpublished document, 1974.

4. Grape, G. R., Model-Based (Intermediate-Level) Computer Vision, Stanford Artificial Intelligence Lab., Memo AIM-201, May 1973.

5. Hollerbach, J. M., Hierarchical Shape Description of Objects by Selection and Modification of Prototypes, MS Thesis, Electrical Engineering Department, Massachusetts Institute of Technology, June 1974.

6. Kuipers, B., A Frame for Frames: Representing Knowledge for Recognition, AI Memo 322, MIT AI Lab., Cambridge, Mass. March 1975.

7. Lozano, T., Parsing Intensity Profiles, AI Memo 329, MIT AI Lab., Cambridge, Mass. May 1975.

8. Marr, D., Analyzing Natural Images: A Computational Theory of Texture Vision, AI Memo 334, MIT AI Lab., Cambridge, Mass. June 1975.

9. Minsky, M. L., A Framework For Representing Knowledge, MIT Artificial Intelligence Lab. AI Memo 306, June 1974.

10. Piaget, J. and Inhelder, B., The Child's Conception of Space, W. W. Norton & Co., New York, 1967.

11. Roberts, L. G., Machine Perception of Three-dimensional Solids, in Optical and Electrooptical Information Processing, Tippett, J. T., et al (ed.). MIT Press, 1965.

12. Schank, R., and Rieger, C., Inference and the Computer Understanding of Natural Language, AIM-197, Computer Science Dept., Stanford University, May 1973.

13. Sutherland, I., Sketchpad: A Man-Machine Graphical Communication System, Tech. Report 296, MIT Lincoln Laboratory, Lexington, Mass., January 1963.

14. Waltz, D., Generating Semantic Descriptions From Drawings of Scenes With Shadows, AI-TR-271, MIT AI Lab., Cambridge, Mass., November, 1972.

15. Winograd, T., Procedures as a Representation for Data in a Computer

Program for Understanding Natural Language, MIT Project MAC TR-84, Cambridge, Mass., February, 1971.

16. Winston, P., Learning Structural Descriptions From Examples, AI-TR-231, MIT AI Lab., Cambridge, Mass., September, 1970.